Xenon Product Requirements Document

Project Title: GitHub Auditor

Team

- **Team name:** So Far, So Good
- **Lead:** Chris Yang
- **Scribe:** Thomas Zhang
- **Other members:** Nico Wong, Edward Thai, Kyle Stubbs

Introduction

Background

Xenon Partners manages a portfolio of many companies and provides them services critical to improving their success. They provide capital and insight to the companies to help them grow and reach the next level. This has resulted in the majority of these clients strategically exiting and making it to the next step.

Problem

Each of the companies that Xenon Partners manages is in charge of a GitHub organization. These companies do not have the tools necessary to thoroughly dive into the repositories and check for potential security issues. Sensitive information that could cause harm within the organization could go undetected if it is not picked up when being pushed into the repository. Xenon's stakeholders would massively benefit from a tool that could automatically detect such issues and have an easy way of accessing and understanding what is going on.

Existing Solution

GitHub already has a built-in dashboard designed to track activity within an organization. This provides great insight into the commits that are occurring within the repositories. However, it does not provide any insight into the security of the organization, which requires manual checking for information such as two-factor authentication or Dependabot alerts. This is effective for looking at the activity that is occurring within the organization, but the lack of sufficient security information makes it inefficient to check for each possible concern.

Goals/objectives

The goal of our project is to implement a GitHub auditor that contains insights into the repositories contained within a company's organization. This will consist of:

- GitHub API server integration to make requests multiple times per day.
- UI Dashboard in order to display information to the permitted users.
- GitHub authentication in order to verify that the users are allowed to access the necessary dashboard.
- Alerts set up to notify users in case of any potential security concerns.

In addition to the dashboard being set up to display all of the information in a user-friendly way, it would be very beneficial to notify the users of any possible security issues, so that way they are notified without having to look at the dashboard. These alerts would report on issues pertaining to:

- Two-factor authentication requirement
- Outside contributors
- Dependabot alerts
- Pull requests lacking an assigned viewer

- Sensitive information such as API keys or passwords in the code

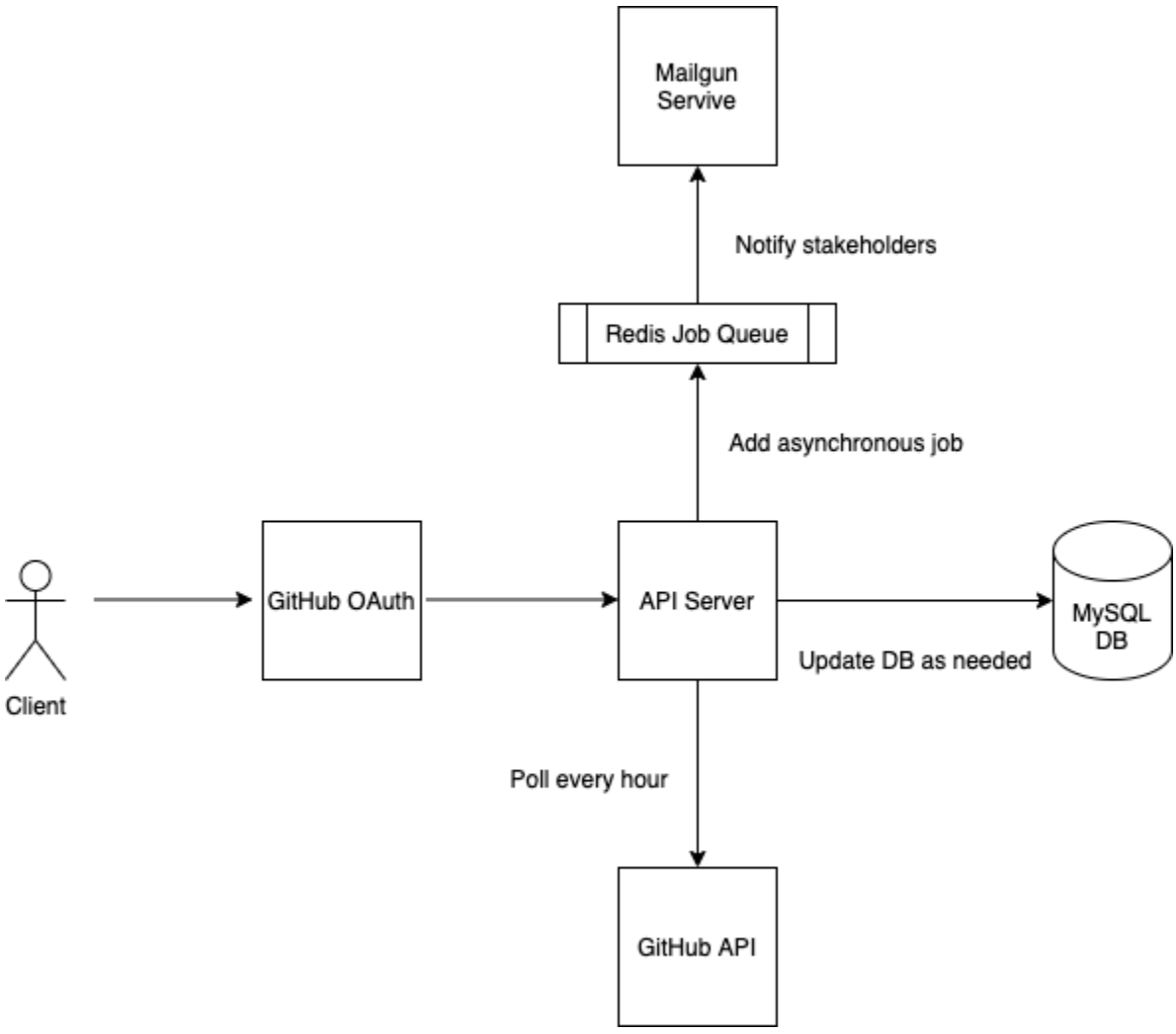
These security concerns will either be detected from the API or use a text-processing tool to detect certain risks in the code. The clients will receive an email containing all of the information needed to alert them, and then the data will be easily accessible when they go to take a look at the dashboard.

Assumptions

- Xenon Partner's clients own a GitHub organization containing multiple repositories.
- Security concerns will happen and need to be addressed
- The preferred method of receiving alerts is by email rather than another form of alert
- The dashboard will be easy to interpret and understand by clearly organizing and labelling everything

System architecture overview

1. Diagram



2. User interaction and design

The client starts off by authenticating themself on the frontend using GitHub OAuth, which will then enable them to make requests to our API server for things like obtaining Dependabot alerts or retrieving two-factor authentication status of certain repositories. The API server will obtain such information by reading from our MySQL database, acting as our primary source of information. We plan to asynchronously keep our database up to date with the actual state of repositories that we are examining via a polling mechanism. Our API server will be polling from GitHub's API once every hour to see if any changes have been made since the last time we updated our database (i.e. detecting if new Dependabot alerts have appeared or if current ones have been resolved). The motivation behind this polling mechanism is intended to avoid the situation where our API servers will have to synchronously make requests to the GitHub API for information, which can lead to added latency for serving responses back to the client. If any such changes are found when polling from GitHub's API, two things will happen. First, we will update our database accordingly to make sure our data is now up to date with what is actually present on GitHub; since we are only checking for updates once every hour, there might be times where our database will contain stale data, but we clarified with our Xenon partners that eventual consistency is sufficient for this system. Second, we will add an asynchronous job to our Redis-backed queue. There will be several workers consuming tasks from the queue, which will then trigger the Mailgun API to notify the relevant stakeholders of something that needs attention. The reasoning behind this queuing model is that we don't want our API server to have to synchronously communicate with the Mailgun API; otherwise, it will have to wait for a response every time it makes a request to the Mailgun Service, which is not scalable.

Requirements

User stories

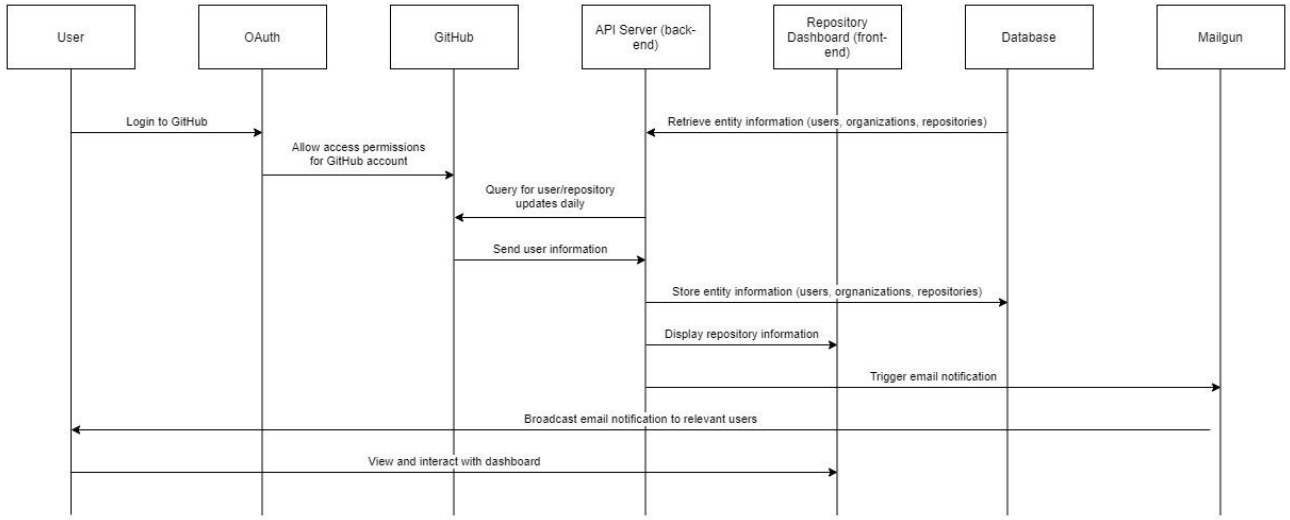
- As a user, I can log in through GitHub so that all my organizations are automatically connected.
- As a user, I can see all my GitHub organizations and choose which ones I want to track.
- As a user, I want a clear, unobstructed view of any security flaws in my organizations, including:
    - Two-factor authentication
    - Outside contributors
    - Dependabot alerts
    - Pull requests lacking an assigned reviewer
    - Sensitive information such as API keys or passwords in the code
- As a user, I can be notified about changes in any of the above security flaws via email.
- As a user, I can click on any of the security flaws to bring up more detailed information.
- As a user, I want this security information to be relatively up-to-date so I have a realistic overview of my organizations' security status.
- As an admin, I have access to all organizations tracked by GitHub Auditor.

Workflow

- GitHub: Feature Branch workflow
    - Feature development takes place in a dedicated branch outside of the main branch.
    - The main branch never contains broken code.
- Local testing

    - ngrok for local testing of remote integrations
- Heroku deployment for staging/production
    - Separate environments for staging and production

System model (sequential)



Appendix

Technologies employed

Ruby on Rails, Redis, GitHub OAuth/API, PostgreSQL, React.js